Test Automation for Regulated Software Systems

This is a booklet in a series from the EUREKA ITEA Testomat Project - The Next Level of Test Automation

Please follow us on:
Twitter: @TestomatProject
YouTube:https://www.youtube.com/testomatproject

Content

1. Introduction

Today's software-intensive systems have assumed a complexity that makes it difficult to foresee all possible situations and contingency that may occur. One reason for this are the ubiquitous connectivity of systems with other systems, their environment or the access to large sets of data that enable reasoning on a much more reliable, but varying basis. Testing such systems is challenging, because testing techniques do not evolve with the same velocity as the complexity of software systems. Automating parts of the testing activities seem to be the only way to tackle the testing challenge adequately. It is important that tests today are automated, so they can be repeated, and as such provide proof of validation in detail for e.g. regulatory purposes. The research and industry best practice has now moved beyond the existing safety-critical standards, which consequently is not being sufficiently updated. In the ITEA 3 TESTOMAT Project we have identified this, and can show that the new methods, not only provides better tested and validate software, but also is possible to achieve with a reasonable cost for the industries, and that these techniques are mature and sufficient to be accepted into the new

standards and regulatory descriptions. We therefore urge you to make sure these results find their way into the right committees,- as we want to save lives in the future by preventing software bugs.

2. How Software Testing benefits from Test Automation

Software testing per se is a highly creative task when it comes to finding the best usage samples (test cases) that will detect a fault if there is one. Writing and executing test cases (potentially over and over again, e.g. in regression testing) manually however, are error prone, tedious and resource consuming tasks. Test automation is generally understood as the automated execution of test cases against the system under test. This is, however, only one testing activity that bears the potential of being automated. Others are test design, test environment setup, test result analysis etc. In particular the activities of the dynamic test process have a great automation potential.

Among others, test automation has the following advantages[1]:

- More tests can be run per build

[1] Taken from ISTQB Test Automation Engineer (TAE) Syllabus

- The possibility to create tests that cannot be done manually (real-time, remote, parallel tests)
- Tests can be more complex
- Tests run faster
- Tests are less subject to operator error
- More effective and efficient use of testing resources
- Quicker feedback regarding software quality
- Improved system reliability (e.g., repeatability, consistency)
- Improved consistency of tests

Of course, test automation comes along with a trade off of increased investment compared to manual testing. Addressed properly, test automation helps deliver better software systems.

Test automation will never be the only way of performing test activities. There are some testing activities that have lower automation potential such as analyzing the test basis. Eventually, at some test levels, manual testing will remain the dominant testing kind such as in acceptance testing. However, these manual testing activities will also benefit from the clear processes that are required for a test strategy that incorporates

test automation. As such, these side-effects are indirect benefits of test automation.

3. Why Test Automation Ought to be Required for Regulated Software

It is not about automating each and every testing activity. As discussed earlier this cannot and will not be the aim. Looking at test automation from a compliance point of view, test automation should be more emphasized in the standards that are relevant for regulated software.

Regulated software is usually required in domains where failures in certain areas are not acceptable. Test automation frees resources that would be bound otherwise in testing. This holds in particular true for regression testing. The freed resources can be further invested in testing to increase the coverage. More coverage means more eligible test results in the end.

For domains where vendors have to prove conformance with technical standards (e.g. implementation of communication protocols and interoperable devices and services thereof), it has proven beneficial for the regulatory agency to also provide a set of standardized automated test cases along with the standardized specifications. The vendors have to run these test cases

against their implementation, before even submitting it to the regulatory agency for deeper analysis. Both sides benefit from such an approach. The vendor can rely on the given standardized test cases to argue that their implementation fulfills a (reduced) set of conformance requirements. The benefits for a regulatory agency are mainly twofold: First, the standardized test cases can be understood as precise, yet executable, requirements specification. In particular for highly critical or very complex requirements it is a good strategy to complement the requirements specification with standardized test cases to resolve possible ambiguities or inconsistencies in the requirements specification. Second, the test result analysis of the regulatory experts, basically a time consuming and costly task, becomes easier and faster, and can even be automated itself. If the standardized test cases also have to follow a standardized, yet structured test log, the regulatory experts do not have to cope with a plethora of tools and test log report formats, but only with their standardized format.

4. Advanced Test Automation Techniques

In the TESTOMAT Project, a number of advanced test automation techniques have been focused that go beyond traditional

automated test execution. In the ubiquitous computer age, software must satisfy numerous quality aspects. Functionality and functional safety is just one category of these aspects. In particular non-functional properties (such as security, performance, interoperability, reliability) of software-intensive systems becomes more and more important, as software systems become more open and connected with each other and the test environment. In the TESTOMAT Project we applied several test automation techniques that have proven helpful and beneficial in producing better (in a sense of test results for both non-regulated and regulated software systems. *Better* has different dimensions here: More efficient in detecting faults (through automated test design), increased coverage of code or requirements (through mutation testing, fuzzing or simulation), faster in test design of meaningful test cases, more frequent test cycles (through CI/CD), simplified test result analysis (through purposeful visualization). All these techniques have been applied on case studies that stem from a regulated domain. The remainder of this section gives a more detailed overview of these techniques.

4.1 New Test Coverage Method: Mutation Testing

Given the widespread reliance on test automation, lots of research went into measuring the quality of test cases. Usually test quality is measured in terms of code coverage, i.e., the proportion of code that is executed by the test suite. The measures most often used in industry are statement coverage and branch coverage[2].

The trial is to make this at a reasonable cost. Today we can show that it is both possible and feasible to improve your coverage with Mutation Testing

[2] [Gopinath2014]

.

Mutation testing means that you change some aspect of the program, data or context, and then repeat your test cases or test suite with the goal that your test suite is complete, and it will detect this change. The change that normally is very small is a variant of your system but can also be called an “amplified” system. If your test does NOT detect this change, the variant software, or amplified system, is then called a mutant. The goal will now be to create a test case that “kills” the mutant, meaning - that makes sure that your new test case (suite) detects this change. This process is then repeated (and automated) over and over through a set of rules. Test suites are measured by the percentage of mutants that

they kill. The whole mutation analysis ultimately results in a score known as the mutation coverage. This is the number of mutants killed divided by the total number of mutants injected.

In 2017, Ramler et al. reported on a second application of mutation testing on an industrial scale, this time for a mechatronic system written in C, comprising 60,000 lines of code[3]. Unit testing was deemed important and 100% MC/DC coverage was set as a goal. From a practical standpoint, the authors confirmed that it is a non-trivial task to integrate the mutation analysis into the development pipeline. To handle the scale of the mutation analysis (a total of 4,071 hours), they parallelized the test execution cycle and distributed it across a cluster of standard desktop PCs. The whole mutation analysis resulted in 27,158 live mutants. They manually sampled 200 of them and 24% of them were classified as equivalent mutants (false positives). The time to manually review each mutant was recorded as well and was quite reasonable: 2 minutes on average and 20 minutes at most.

The authors conclude that a mutation analysis provides actionable suggestions for the test engineers. "… mutation testing provides hints about deficiencies in test cases that are otherwise hard to discover. The feedback can

[3] [Ramler2017]

be directly used for revising and enhancing the tests.” [4].

4.2 Simulation for Better Testing

A final result for safety-critical testing is to test sufficiently in the real environment. For many safety-critical systems, the real environment is costly to set up, and vulnerable to bugs. Instead, testing can be done using a simulated or emulated environment. All software can in fact be tested thoroughly “in vitro”, and the fact is, in simulation one can sometimes test more extreme situations than can be done in real. Most of today’s software systems are distributed or connected with other systems, devices or the environment. For testing, representative test environments need to be established, however, the more complex the real environment becomes, the more expensive will the mimicking test environments be. Furthermore, the sheer number of varying test environments might be so numerous that it is not feasible to address them all. Simulation finds a remedy to this dilemma.

Simulation is also well suited for fault injection in the course of robustness testing or disaster recovery testing. In a simulation the worst case, that is the occurrence of a

[4] [Ramler2017]

hazard, can be tested and how the system under test may recover from that worst case, if ever. Faults external to the system under test are often hard or even impossible to establish manually. For example, how shall a tester manually introduce an error into the driver software of a hard disk that prevents the system from accessing the hard disk? Waiting for a real driver failure would be way too inefficient, so having a simulation in place enables us to let external failures occur as often as required.

Another very new topic is virtual commissioning of systems prior to release. Virtual commissioning is based on the idea of digital twins (or digital shadows), where the real code of systems, sub-systems or devices (such as robots, vehicles etc.) are deployed to a simulated representative of those systems, sub-systems or devices. As a result, the integrated system is subject to testing, before the real system has to be built or set up. Contingencies, robustness, or critical situations can be fully explored on the basis of a simulation that mimics the reality as close as possible. In the TESTOMAT Project, simulation has been applied to many use cases from different domains such as automotive, aerospace or industry 4.0 with great success.

4.3 Fuzzing

Fuzzing is a technique that stimulates the test item with invalid or unexpected inputs aiming at discovering weaknesses that result from missing or faulty input validation. Inherent to fuzzing is the high degree of automation from its origin where input data were generated randomly and submitted to the SUT completely automatically, including checks for potential crashes.

Over the years, fuzzing has been evolved along the following dimensions:

- Test basis, i.e. from nearly no test basis with random fuzzing, to specification-based fuzzing and fuzzing the source code
- Fuzzing heuristics, to mutate inputs for the test item, or to mutate specifications and generate input data from the mutated specification
- Syntax to semantic, to fuzz on a binary level or syntactic level, i.e. fuzzing with respect to input data and the semantic level that comprises dependencies between fields of the same inputs, data flows, states, and messages flows.

Although fuzzing is mainly known and understood as an automated security

testing technique, it is applicable to fault tolerance testing of safety-critical systems.

4.4 Continuous Integration (CI)/ Continuous Delivery (CD)

Modern software development is centered around quickly developed small and incremental steps. This is opposed to the V-model where the software artifacts are developed in different phases. CI/CD targets quicker response cycles after changes have been made to the code bassis. Testing, building and deploying the system after changes enable the vendor to identify issues with the last changes soon.. Ideally on the very same day, the change has been introduced. This leads to much more reliable delivery of the software system to be developed.

Since the compliance guidelines are around for longer periods (decades instead of years), they are based on a more waterfall development. The (technical) delivery of a functional correct software artifact (including all the required test-steps, often performed automatically) in a continuous flow will be followed by a compliance phase before the product can be actually delivered and put on the market.

These are the formal compliance requirements that impact the validation phases in the CI/CD process. And they might slow down the promise of the CI/CD. Nonetheless, CI/CD has been proven successful in the development of complex software systems and should be considered even for regulated software.

5. Steps to take to put a safety-critical product on the market (when it comes to testing)[5]

There is some difference in putting a safety-critical product than a normal product on the market, even if the majority is very similar. We will only highlight some aspects to consider, w.r.t. testing, validation and certification. First, it is important to know that certification adds cost to a product, and

[5] *This chapter is written mainly for people in software/systems/testing that previously not have addressed safety-critical products and marketing, e.g. in the context of IoT.*

that you need specific knowledge of what it takes to get there. It is important to understand what regulatory bodies you work with, and that you need - from the start of your development - to work with certified tools, certified partners and both plan and conduct a series of important verification and validation aspects to secure your software. Note that each domain often has its own set of standards.

Standards are introduced for a long period of time, and focus on the process of producing the safety critical system. There is a tension between the use of modern (testing) techniques as for example introduced by TESTOMAT Project and using (only) the techniques that are mandatory in the standard. We advocate that modern techniques are in no conflict with creating safety-critical software, but instead increases the repeatability through automation, and as such increases the security. If there are less "human-in-the loop"-construction, the entire process can be made more reliable. We also advocate the latest of advanced verification and test techniques, that we can show brings value to the software development process, e.g. utilizing mutation testing to complement the software unit testing, as well as adding simulations to increase the testing and act as a complement to testing with the

environment. The project aim is to make modern technologies cost-efficient to use and complement traditional (often manual and slow) approaches to testing.

The following subsections describe three different safety-critical domains, where the standards are on the verge of requiring updates with respect to testing techniques because of modern software systems. The examples stem from the eHealth domain (medical devices), automotive domain (ISO[6] 26262) and aerospace domain (RTCA[7] DO-178B/C).

5.1 eHealth/Medical Devices

The FDA has started to introduce cyber-security concerns into their guidelines in 2018. It started to introduce mandatory software testing techniques. We quote the FDA[8] document: "Content of Premarket

[6] International Organization for Standardization: https://www.iso.org/

[7] Radio Technical Commission for Aeronautics: https://www.rtca.org/

[8] FDA draft guideline: https://www.fda.gov/regulatory-information/search-fda-guidance-documents/content-premarket-submissions-management-cybersecurity-medical-devices

Submissions for Management of Cybersecurity in Medical Devices":

A description of the testing that was done to ensure the adequacy of cybersecurity risk controls (e.g., security effectiveness in enforcing the specified security policy, performance for required traffic conditions, stability and reliability as appropriate). Test reports should include:

(a) testing of device performance
(b) evidence of security effectiveness of third-party OTS software in the system.
(c) static and dynamic code analysis including testing for credentials that are "hardcoded", default, easily-guessed, and easily compromised.
(d) vulnerability scanning
(e) robustness testing, which is often helped by technologies like fault injection and mutation testing.
(f) boundary analysis, complemented with data-flow analysis. (and random data testing)
(g) penetration testing
(h) Third Party test reports

It is an example where specific test technologies are mandatory practice to become compliant.

A similar development happened in the EU, where the “Guidance on Cybersecurity for medical devices”[9] that focussed on security related testing such as the test automation technique fuzzing:

“The primary means of security verification and validation is testing. Methods can include security feature testing, fuzz testing, vulnerability scanning and penetration testing. Additional security testing can be done by using tools for secure code analysis and tools that scan for open source code and libraries used in the product, to identify components with known issues.”

This is an example where testing techniques are becoming mandatory as part of the certification process.

5.2 Automotive Functional Safety

The goal of Functional Safety (FuSi) is to reduce the risk from electrical and electronic (E/E) systems to a tolerable level. Especially with regard to highly automated and autonomous driving, the Functional Security of E/E systems is increasingly important. The safe development of these systems according to an established

[9] MDCG 2019-16 - Guidance on Cybersecurity for medical devices: https://ec.europa.eu/docsroom/documents/41863

standard minimizes the occurrence of random and systematic errors, thus preventing personal injury and damage to property.

When developing and testing your systems and or components, teams follow the automotive-specific standard ISO 26262: 2018, which refers to the current state of the art and includes guidelines for the design, development, series and post-series support of E/E systems in the field of cars, trucks, buses and motorcycles.
The ISO 26262 defines standards for the safety lifecycle of individual automotive products for the

- concept phase,
- product development at system, hardware and software level,
- production and operation and
- service and decommissioning.

The ISO 26262 also provides an automotive-specific risk-based approach for determining risk classes or ASILs:

- identifying and assessing safety risks by Severity, Exposure and Controllability Classifications,
- establishing requirements to reduce those risks to acceptable levels and
- tracking requirements to ensure that an acceptable level of safety is achieved in the delivered product.

5.3 Aerospace Safety-Critical Systems

Regarding the *RTCA*[10] *DO-178, Software Considerations in Airborne Systems and Equipment Certification* and its European equivalent *EUROCAE*[11] *ED-12C*, it is the primary document that it is used by the regulatory aerospace agencies, like FAA[12] or EASA[13], for approving software-based aerospace systems. The considerations of the standard are intended to support the objectives, according to the Software Level or Design Assurance Level or DAL, that are determined from the safety assessment process and hazard analysis by examining the effects of a failure condition in the system. The failure conditions are categorized by their effects on the aircraft, crew, and passengers as:

- Level A: Catastrophic.
- Level B: Hazardous.
- Level C: Major.
- Level D: Minor.
- Level E: No Effect.

[10] Radio Technical Commission for Aeronautics: https://www.rtca.org/
[11] European Organisation for Civil Aviation Equipment: https://www.eurocae.net/
[12] Federal Aviation Administration: https://www.faa.gov/
[13] European Safety Agency: https://www.easa.europa.eu/

The following processes have to be followed in order to be compliant with the DO-178:

- Planning,
- Development,
- Verification,
- Configuration Management,
- Quality Assurance and
- Certification Liaison.

In the last version of the standard (DO-178C) a new chapter has been introduced with guidelines for the Model Based Development and Verification.

The example of DO-178C is that the required safety provided could be considered old fashioned when it comes to some of the verification technologies. The main claim is that new automation, simulation and new technologies have been validated as better than the existing directives.

5.4 Conclusion and a Plea!

We, the TESTOMAT project, therefore urge the use of more updated and modern technologies - as well as we urge the regulatory bodies to update their standard. In particular we suggest increasing the use of automation and mutation testing as well as more exhaustive use of varied simulation scenarios.

6. Examples from TESTOMAT Project on testing Safety-Critical Software

The following subsections summarize success stories of the TESTOMAT Project case studies, where advanced test automation techniques were successfully applied to software systems from safety-/ security-critical domains.

6.1 Testing Trains's in Simulation

In this section we describe the importance of simulated environments as a base for test automation. Validation of software developed for safety critical functions on trains includes testing to show that the failure modes are working correctly. The train shall work in a safe way even if a part of the train starts to work incorrectly. It is difficult to test failure modes on a train without causing damage to the tested train. Therefore, a simulated environment that is surrounding the system under test, is necessary to perform nondestructive testing of failure modes. This leads to the necessity to qualify the simulated environment to show evidence that it satisfies all mandatory regulatory requirements for the environment to be considered as a trusted test tool. A qualified simulated test environment makes it possible to run both testing of failure

modes and, as an additional possibility, to automatically run frequent and extensive automatic software regression testing.

In the same way as the qualification of the simulated environment is done it is possible to qualify an emulated runtime environment, for the system under test, to be a trusted test tool. The complete distributed control system hardware can be replaced with software functions if the virtual environment reaches the state of a qualified test tool. This virtual system with the complete application software running on it will then, at a low cost, be available for all developers and testers in the project as soon as the environment is installed on their standard laptops. The availability of this virtual test environment is a necessary base for increasing the use of automatic testing on all test levels from unit tests to integration and acceptance tests. The work to create the emulated runtime environment for the Train Control System has to a large extent been done in the Testomat Project.

The emulated runtime environment is also a condition for automatic regression testing of different train configurations. Train units can be coupled together in different patterns to form the requested train and the number of possible train configurations increases rapidly with the number of train units

engaged when coupling the train together. The only way to efficiently regression test a huge number of train configurations is to use a simulated environment with the emulated runtime environment together with automatic test generation methods. Different possible ways to realize this were investigated in the TestomatProject.

The creation of simulated and emulated environments used to enable early software testing is a time and money saving factor when the objective is to efficiently develop qualitative and safe software for trains.

6.2 Model-based testing and SafeScrum in safety-critical Rail Software Development

The development of safety critical rail software is internationally regulated by CENELEC standards, such as CENELEC EN 50126, EN 50128, and EN 50129. These regulations encourage safety-related projects traditionally to develop in a waterfall manner. Our experience with iterative software development methods, such as Agile and Scrum is that these are a good approach to create high quality software. The incremental nature allows us to learn in the project and to fix mistakes in a fast and

efficient way. We were looking for a way to combine Agile and safety critical. We have found this in a combination of model-based testing and SafeScrum. SafeScrum is an iterative software development method that is developed by SINTEF and NTNU which is approved for safety critical systems. Model-based testing is a low code scriptless test automation solution. It allows for thorough and fast testing. Low code models are created incrementally and are extended every sprint. Tests are automatically generated from the models and executed against the System. Because no test scripts need to be programmed (they are generated from the low-code models) it is possible to test with a big test-set in a maintainable manner. Together with InTraffic and ProRail we use this approach to develop safety critical systems.

6.3 Simulation-based testing of safety-critical systems comprising rare events

Due to the constantly increasing interconnection of control units and assistance systems within a car, the effort required to prove their function for correctness is also increasing, especially with regard to the focus on certification. Simple methods such as testing of corner cases and equivalence classes are rapidly reaching their limits and are no longer

up-to-date. Simulation-based approaches, which use intelligent algorithms to analyze complex models and identify rare events, have reached product maturity. By means of statistical estimation, they can provide reliable information on residual risks, and thus make a valuable contribution to reducing safety-critical events. Within the TESTOMAT Project, for instance, OFFIS conducted a case study on an industrial use case provided by AKKA Technologies where we applied such algorithms on a battery management system. In that case study we showed that these algorithms are able to identify critical parameter combinations much more efficiently compared to traditional Monte Carlo simulation methods. That is, critical events are detected with increased probability while at the same time much less simulation budget is required therefore. As the underlying methodology is rather general, the approach can be easily adapted to other industrial use cases as well.

6.4 Fuzzing of financial application

Kuveyt Turk Bank applied fuzzing with the Fuzzino tool of Fraunhofer FOKUS to its financial application case study. The integrated Fuzzino into the test environment and executed fuzzed functional test cases

against the financial application. Fuzzino resolved two problems in that case study:

1. Reuse of existing functional test cases and test environment for security testing
2. Automated prioritization of security test cases based on type-specific security heuristics.

Fuzzing (with Fuzzino) was deemed very helpful in conducting automated security testing of the financial application with little, respectively adequate manual resource for setup and execution of security test cases.

6.5 Why Mutation testing gives us better coverage than current standard for aeroplanes

Saab made the strategic decision to leave the academic problem with equivalent mutants behind, and focus on a user-centered approach in an industrial setting. In parallel with our insight growing of how to use mutation testing together with the results from Google in their papers from 2017 and 2018 we came to the conclusion that by changing the problem to what the user is actually requesting, insight into the test cases and the program under test,

equivalent mutants could be reduced from a roadblock issue to a nuisance.

By exploring the relationship between a mutant and which test cases that killed it, we noticed that it could be exploited to create user insight. This relationship could be used to answer key user questions, such as:

- *"Can the technique tell me what is unique about a test case because I have this test case that I have a feeling is good but I want objective facts to tell me what it is that is actually good about it?"*
- *"I have these test cases that I wrote during one phase of the development. I haven't really kept them up to date. Since then I have written many more test cases. I wonder if these old test cases are worth keeping or if they can be thrown away?"*
- *“I wonder if all test cases actually verify my software”*

By answering the users actual questions it means that tool and technique is useful in practise. This is the most significant insight for Saab and the avionics domain that we have learned through the Testomat Project.

We are certain that there are more interesting ways of exploiting this relationship between the test cases and the mutants than what we have found so far. This shift in the focus means that we have a clearer road ahead of how to align mutation testing with the avionics safety standard RTCA/DO-178C.

The relationship can be used to find some of the faults that previously could only be found by manual inspection of the test cases. It may never be able to fully replace a manual inspection but the technique can be used to automatically find severe faults in the test suite at an early stage in the process with minimal effort where it is *cheap* to correct the mistakes. It is an automated sanity check of a test suite. If the test suite passes the sanity check it is worthy of a costly manual inspection.

Summary - A Final Word to Regulatory Agencies

The TESTOMAT-Project has created this booklet to make regulatory agencies aware of the constant progress in technology and methods in the realm of test automation. Many of the project partners stem from a regulated domain and work on regulated software systems.

TESTOMAT dealt with advanced test automation techniques. The techniques described in this booklet have been applied to real case studies not only but also from regulated domains.

As one of our major results in the TESTOMAT Project we have tried mutation testing out in industrial settings. As a result we can today recommend mutation testing to be a mandatory technique for all safety-critical software. The cost and effort is reasonable compared to the improved quality. We can improve quality and should use mutation test analysis for all safety-critical software. And if you are mature enough and fulfill the criteria for mutation testing, it will bring better quality

We, the TESTOMAT Project, ask the regulatory agencies to have a closer look on test automation in general, and the advanced test automation techniques in particular. Our overall ambition is to help the regulatory agencies in keeping their relevant standards and processes up to date, so that in the end, regulated software systems of higher quality and less defects will be produced.

If you have any further questions, please contact us.

Acknowledgements:
This booklet is produced by
EUREKA ITEA3 TESTOMAT PROJECT
The Next Level of Test Automation
Find out about us on the web:
https://www.testomatproject.eu/
Follow us on Twitter
@Testomatproject

You can also watch us on Youtube
https://www.youtube.com/testomatproject

This booklet was produced by a research collaboration between the following partners:

The Testomat Project is sponsored by:

Disclaimer: The content of this booklet is true to the best of our current knowledge. The authors, publishers, participating partners of the project as well as the funding agencies disclaim any liability in connection with the use of this information.